/semgrep-rule-variant-creator

Source: `~/.claude/skills/tob-semgrep-rule-variant-creator/skills/semgrep-rule-variant-creator/SKILL.md`

---

name: semgrep-rule-variant-creator description: Creates language variants of existing Semgrep rules. Use when porting a Semgrep rule to specified target languages. Takes an existing rule and target languages as input, produces independent rule+test directories for each language. allowed-tools:

- Bash
- Read
- Write
- Edit
- Glob
- Grep
- WebFetch

Semgrep Rule Variant Creator

Port existing Semgrep rules to new target languages with proper applicability analysis and test-driven validation.

When to Use

Ideal scenarios:

- Porting an existing Semgrep rule to one or more target languages

- Creating language-specific variants of a universal vulnerability pattern
- Expanding rule coverage across a polyglot codebase
- Translating rules between languages with equivalent constructs

When NOT to Use

Do NOT use this skill for:

- Creating a new Semgrep rule from scratch (use `semgrep-rule-creator` instead)
- Running existing rules against code
- Languages where the vulnerability pattern fundamentally doesn't apply
- Minor syntax variations within the same language

Input Specification

This skill requires:

1. **Existing Semgrep rule** - YAML file path or YAML rule content
2. **Target languages** - One or more languages to port to (e.g., "Golang and Java")

Output Specification

For each applicable target language, produces:

```
<original-rule-id>-<language>/
├── <original-rule-id>-<language>.yaml     # Ported Semgrep rule
└── <original-rule-id>-<language>.<ext>    # Test file with annotations
```

Example output for porting `sql-injection` to Go and Java:

```
sql-injection-golang/
├── sql-injection-golang.yaml
└── sql-injection-golang.go

sql-injection-java/
├── sql-injection-java.yaml
└── sql-injection-java.java
```

Rationalizations to Reject

When porting Semgrep rules, reject these common shortcuts:

Rationalization	Why It Fails	Correct Approach
"Pattern structure is identical"	Different ASTs across languages	Always dump AST for target language
"Same vulnerability, same detection"	Data flow differs between languages	Analyze target language idioms
"Rule doesn't need tests since original worked"	Language edge cases differ	Write NEW test cases for target
"Skip applicability - it obviously applies"	Some patterns are language-specific	Complete applicability analysis first
"I'll create all variants then test"	Errors compound, hard to debug	Complete full cycle per language
"Library equivalent is close enough"	Surface similarity hides differences	Verify API semantics match
"Just translate the syntax 1:1"	Languages have different idioms	Research target language patterns

Strictness Level

This workflow is **strict** - do not skip steps:

- **Applicability analysis is mandatory**: Don't assume patterns translate
- **Each language is independent**: Complete full cycle before moving to next
- **Test-first for each variant**: Never write a rule without test cases
- **100% test pass required**: "Most tests pass" is not acceptable

Overview

This skill guides the creation of language-specific variants of existing Semgrep rules. Each target language goes through an independent 4-phase cycle:

```
FOR EACH target language:
  Phase 1: Applicability Analysis → Verdict
  Phase 2: Test Creation (Test-First)
  Phase 3: Rule Creation
  Phase 4: Validation
  (Complete full cycle before moving to next language)
```

Foundational Knowledge

The `semgrep-rule-creator` skill is the authoritative reference for Semgrep rule creation fundamentals. While this skill focuses on porting existing rules to new languages, the core principles of writing quality rules remain the same.

Consult `semgrep-rule-creator` for guidance on:

- **When to use taint mode vs pattern matching** - Choosing the right approach for the vulnerability type
- **Test-first methodology** - Why tests come before rules and how to write effective test cases
- **Anti-patterns to avoid** - Common mistakes like overly broad or overly specific patterns
- **Iterating until tests pass** - The validation loop and debugging techniques
- **Rule optimization** - Removing redundant patterns after tests pass

When porting a rule, you're applying these same principles in a new language context. If uncertain about rule structure or approach, refer to `semgrep-rule-creator` first.

Four-Phase Workflow

Phase 1: Applicability Analysis

Before porting, determine if the pattern applies to the target language.

Analysis criteria:

1. Does the vulnerability class exist in the target language?
2. Does an equivalent construct exist (function, pattern, library)?
3. Are the semantics similar enough for meaningful detection?

Verdict options:

- `APPLICABLE` → Proceed with variant creation
- `APPLICABLE_WITH_ADAPTATION` → Proceed but significant changes needed
- `NOT_APPLICABLE` → Skip this language, document why

See applicability-analysis.md for detailed guidance.

Phase 2: Test Creation (Test-First)

Always write tests before the rule.

Create test file with target language idioms:

- Minimum 2 vulnerable cases (`ruleid:`)
- Minimum 2 safe cases (`ok:`)
- Include language-specific edge cases

```
// ruleid: sql-injection-golang
db.Query("SELECT * FROM users WHERE id = " + userInput)

// ok: sql-injection-golang
db.Query("SELECT * FROM users WHERE id = ?", userInput)
```

Phase 3: Rule Creation

1. **Analyze AST**: `semgrep --dump-ast -l <lang> test-file`
2. **Translate patterns** to target language syntax
3. **Update metadata**: language key, message, rule ID
4. **Adapt for idioms**: Handle language-specific constructs

See language-syntax-guide.md for translation guidance.

Phase 4: Validation

```
# Validate YAML
semgrep --validate --config rule.yaml

# Run tests
semgrep --test --config rule.yaml test-file
```

Checkpoint: Output MUST show `All tests passed`.

For taint rule debugging:

```
semgrep --dataflow-traces -f rule.yaml test-file
```

See workflow.md for detailed workflow and troubleshooting.

Quick Reference

Task	Command
Run tests	`semgrep --test --config rule.yaml test-file`
Validate YAML	`semgrep --validate --config rule.yaml`
Dump AST	`semgrep --dump-ast -l <lang> <file>`
Debug taint flow	`semgrep --dataflow-traces -f rule.yaml file`

Key Differences from Rule Creation

Aspect	semgrep-rule-creator	This skill
Input	Bug pattern description	Existing rule + target languages
Output	Single rule+test	Multiple rule+test directories
Workflow	Single creation cycle	Independent cycle per language
Phase 1	Problem analysis	Applicability analysis per language
Library research	Always relevant	Optional (when original uses libraries)

Documentation

REQUIRED: Before porting rules, read relevant Semgrep documentation:

- Rule Syntax - YAML structure and operators
- Pattern Syntax - Pattern matching and metavariables
- Pattern Examples - Per-language pattern references
- Testing Rules - Testing annotations
- Trail of Bits Testing Handbook - Advanced patterns

Next Steps

- For applicability analysis guidance, see applicability-analysis.md
- For language translation guidance, see language-syntax-guide.md
- For detailed workflow and examples, see workflow.md

Revision #4
Created 2026-02-18 08:40:08 UTC by John
Updated 2026-05-31 20:01:52 UTC by John